AKAMAI WHITE PAPER

Akamai
FASTER FORWARD

Strategies for API Security

INTRODUCTION

The explosion in consumer mobile adoption, computerization of goods and services, and an increase in data generation have driven a change in the way Internet-based businesses are built and consumed. The digital economy has prompted online organizations to facilitate the creation and exchange of information to new channels, partners, and developers with the goal of unlocking new business value for these consumers. Today, connecting businesses to their cross-channel customers is technically driven by Application Programming Interfaces, or APIs.

This white paper explores strategies for protecting APIs by first introducing how APIs are designed, and how similarities between web applications and APIs mark these endpoints as added targets for web attackers. This white paper will also outline the most common types of cyberattacks and concludes with a discussion on Akamai's solution against API exploits.

API DEFINITION

An Application Programming Interface (API) is a contract for how two pieces of software talk to each other. An API can be thought of as a restaurant menu – it dictates how information is delivered from a host and consumed by a client. APIs, however, pose risks beyond traditional web applications because of their required visibility to the end user. By definition, APIs provide transparency to the internal structure of applications and provide granular access to an application's backend, making online businesses more vulnerable than ever before to cyberattack threats.

API designers tend to land on one of two language-neutral messaging formats for exchanging data between their servers and client developers: SOAP and XML. SOAP uses XML as a container for the messages to be exchanged, and REST (RESTful) APIs commonly use JavaScript Object Notation (JSON). RESTful APIs are now the most popular and commonly adopted APIs due to their implementation of standard HTTP methods to retrieve and manipulate resources. JSON is the format of choice for RESTful-based APIs because they are simple and easy to read by humans and easy to parse by computers.

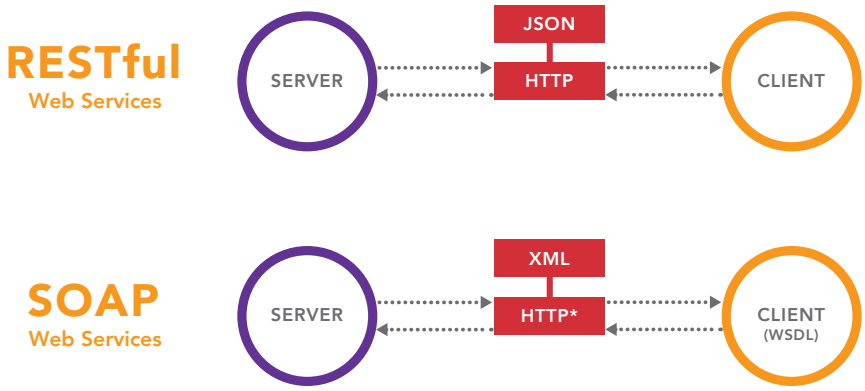

Figure 1 - Prevalent technologies related to APIs

It is important to understand that APIs are built for computer consumption rather than direct client consumption. Many developers make the mistake of assuming that since the API is masked behind another service (ex. Mobile application), security can be managed on the client side and relaxed on the API server.

COMMON CYBER THREATS

Before exploring common API exploits, it is important to understand the lifecycle of an API request. Keep in mind that RESTful API design means that developers use standard HTTP methods to retrieve and manipulate resources, so typical web application vulnerabilities also apply to API requests. Attacks against API endpoints also require less sophistication since APIs, by design, are well documented and open to consumers.

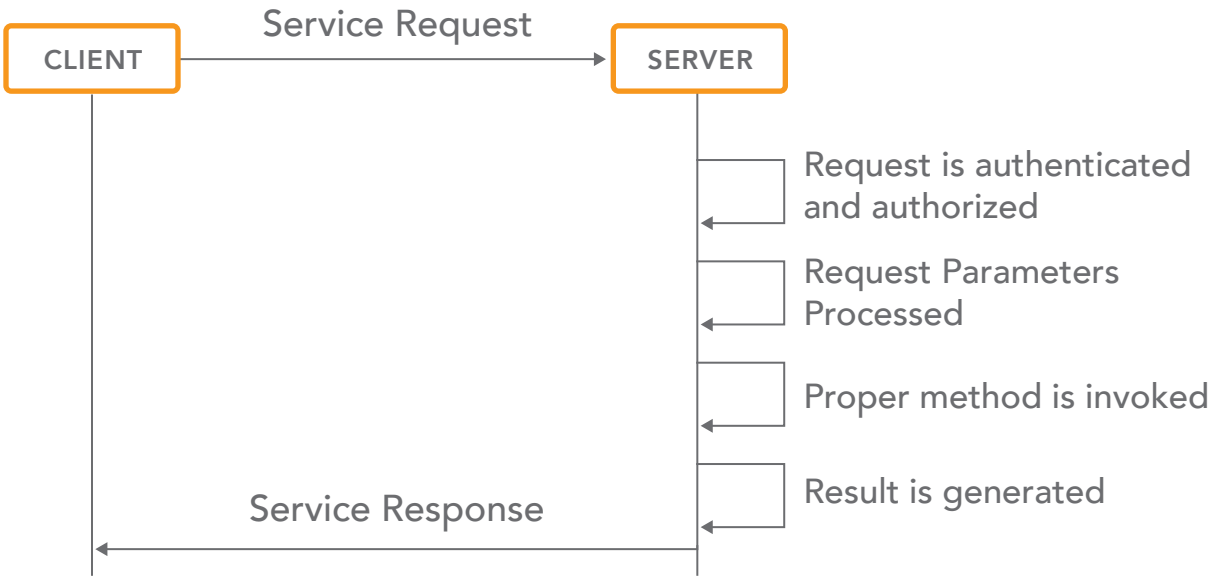

Figure 2 - API lifecycle between client request and server response

For reference, we will use Akamai's Network List Management API listed on the Akamai Developer site.

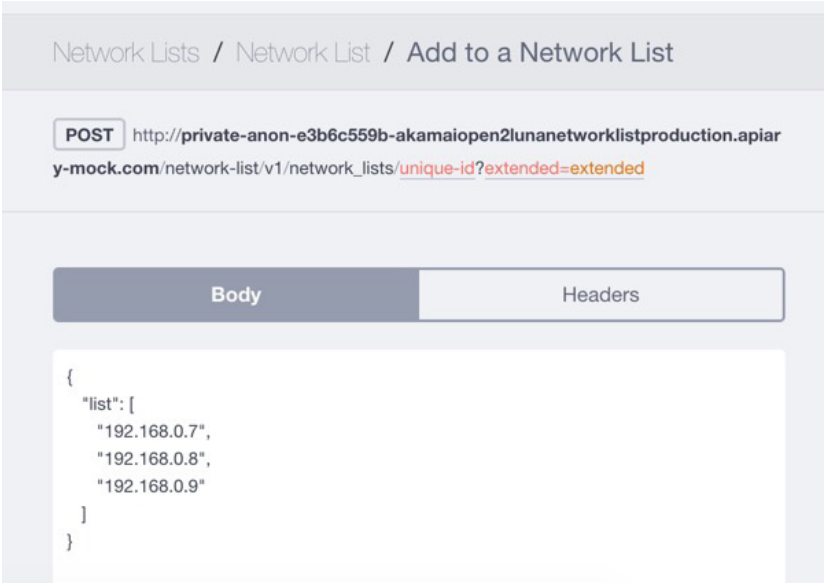

Figure 3 - Create a Network List API call

Application Downtime due to an Excessive Rate of API Calls

The first cyber threat associated to the API lifecycle is exploited during the initial service request between a client and server - a Denial-of-Service attack against the API endpoint. Denial-of-Service (DOS) or Distributed Denial-of-Service (DDoS) attacks can be accomplished by making too many API calls to an application server at any given period or by making slow POST requests. A common security oversight is to protect common web domains but miss adding web application protections against an API domain. In this example, we may have slow POST protection and rate controls against www.apiary-mock.com but not against private-anon-8c8197c799-akamaiopen2lunanetworklistproduction.apiary-mock.com.

Data Theft via MITM Attacks

Another exploitable vulnerability during the initial request is a Man-in-the-Middle attack (MITM). A hacker who successfully performs a MITM attack is capable of intercepting API transactions and revealing or altering confidential information.

A popular example of an MITM attack is the Snapchat Security Advisory documented by Gibson Security in 2013. Gibson Security researchers discovered that Snapchat was utilizing symmetric-key encryption to encrypt snaps between client and server. Symmetric key encryption is a method of scrambling data by utilizing the same key to both encrypt and decrypt data. The "secret" key was the exact same key across all iOS and Android app users and was stored on the application in an easy-to-find manner.

The standard today is to utilize asymmetric encryption, where different keys are utilized to encrypt and decrypt. With proper encryption, data in transit can be better protected from MITM attempts.

Weak Authentication & Authorization

Authentication is the process of understanding who the client is, while authorization is knowing what a given client can do. Making API calls without proper authentication, authorization, and session tracking can allow attackers to take prohibited actions on behalf of someone else. For example, if Akamai's Network List Management API poorly implemented authentication, any hacker could pretend to be a customer and update blacklists and whitelists for a customer's Web Application Firewall (WAF). If an end user was given improper authorization, a hacker could take over the account and have access to the Network List Management API when that level of control was not allowed for that client.

A good example of the risk of ignoring authentication/authorization and allowing clients to make anonymous requests is documented by Troy Hunt on the discovery of controlling vehicle features of a Nissan LEAF in early 2016. Researchers found that the Nissan LEAF API could be exploited to control vehicle features because the Nissan LEAF app wasn't authenticating users.

Figure 4 - Nissan LEAF API authentication vulnerability

The Nissan app was utilizing car VIN numbers to specify how to control the car rather than a username and password. An attacker could cycle through VIN numbers and activate/deactivate air-conditioning/heating features of the car. A second failure was that when submitting a VIN number, attackers also have access to the userID associated to the VIN number, which is a combination of the first and last name of the owner of the vehicle.

Exploiting API Parameters

A significant portion of API vulnerabilities are exploited during the processing of API parameters. API parameters include URL, query parameters, HTTP headers, and POST body. As mentioned above, RESTful APIs leverage standard HTTP methods to retrieve and manipulate resources, so typical web application vulnerabilities apply to APIs. We can examine the similarities of an attack against an API with that of an attack against a web application in the table below. The hostnames, parameters, raw requests, and attack payloads were collected from Akamai's Cloud Security Intelligence database.

Attack Type	Hostname	Parameter	Raw Request
Directory Traversal	openapi.[redacted].com	Path	/[redacted]=../../../../../../../../../../../../../../../../etc/passwd/
Command Injection	www.[redacted].com	Path	/[redacted]/../../../../../../../../../../winnt/system32/cmd.exe?dir/search.json=orgUnitId=E000124
SQL Injection	store.[redacted].com	Query	\?[redacted]=999999.9 union all select 0x31303235343830303536,0x3130323534383030303536,0x313032353438303030536,0x31303235343830303536,0x31303235343830303536,0x31303235343830303536,(select concat(0x7e,0x27,pyg_member.address,0x27,0x7e) from `pyginthg_pyg`.pyg_member Order by uid limit 5,1),0x31303235343830303536,0x3130323534383030303536,0x313032353438303030536,0x31303235343830303536,0x3130323534383030303536,0x31303235343830303536,0x31303235343830303536,0x3130323534383030303536,0x313032353438303035536,0x31303235343830
SQL Injection	store.[redacted].jp	Path	/[redacted]/CHAR(119)+CHAR(104)+CHAR(115)+CHAR(83)+CHAR(81)+CHAR(76)+CHAR(105)
SQL Injection	www.[redacted].com	Body	{"[redacted]":"[redacted]:'4' UNION SELECT 1,@@version—", "[redacted]"}
Cross Site Scripting	api.[redacted].com	Query	/?[redacted]=<script>alert(document.cookie)</script>

Figure 5 - Table of attack types and parameters exploited

There are two mitigation strategies to deal with when fighting parameter attacks:

- **Negative Security Model:** In the negative security model we compare API parameters against a set of blacklisted content to filter malformed or malicious requests directed at the API server. This model applies security rules to mitigate attacks similar to what we see in Figure 5 above – applying rules to fight Layer-7 vulnerabilities such as SQLi, XSS, etc.
- **Positive Security Model:** Alongside a negative security model to filter malformed or malicious requests directed at the API server, requests also undergo a positive security model to validate API parameters against expected values, such as:
  - » Parameter Type (Integer, String, Boolean, etc.)
  - » Maximum value for any integer value
  - » Maximum length for any string value
  - » Maximum number of JSON keys/XML elements
  - » Maximum length for any key/element name
  - » Maximum body size

In our Network List Managed example, hackers could run a DoS attack by adding an unlimited number of IP addresses if Akamai wasn't limiting the maximum number of JSON elements or maximum body size. It's important to note that this is not a typical application exploit such as a SQL injection – a positive security model must be used to mitigate parameter-abuse type attacks.

Privacy can also be subverted if an API server isn't validating required JSON elements. In the Snapchat Security Advisory mentioned above, Gibsonsec also revealed that Snapchat's "friend discovery" API wasn't validating input parameters to the API. The discovery API connected to an end-user's contacts list from the mobile address book and submitted a phone number and name to the API server. The result of the API is a person's Snapchat name and real name. Gibsonsec found that Snapchat wasn't limiting the upper bound of requests that could be made to the API server, and hackers could submit 75,000 phone numbers and receive a Snapchat username to real name to phone number mapping. Hackers later used this same exploit, leading to "SnapchatDB", where hackers leaked 4.6 million phone numbers and names of Snapchat users.

HOW CAN AKAMAI HELP?

Akamai has introduced new workflows for security professionals to protect both JSON and XML-based API endpoints from cybercriminals. Akamai has deployed a three-tiered approach to securing APIs: a positive security model by defining how the API should be consumed, a negative security model by protecting API endpoints from Layer-7 web application vulnerabilities, and API-specific reporting for increased visibility into how end users interact with available APIs.

Figure 6 - Kona Site Defender API Workflow

By defining an API within the Akamai cloud, Kona Site Defender (KSD) can enforce a client request throughout the API request lifecycle, going beyond applying the Kona Rule Set to APIs. KSD now focuses on all possible exploits against APIs including DOS attacks, MITM attacks, authentication and authorization bypass attempts, the exploitation of web vulnerabilities, and the exploitation of API parameters.